Facultad
de
Ciencias

CLASIFICACIÓN DE IMÁGENES MÉDICAS DEL PACS DEL HOSPITAL UNIVERSITARIO MARQUÉS DE VALDECILLA CON UNA RED NEURONAL CONVOLUCIONAL

(CLASSIFICATION OF MEDICAL IMAGES FROM MARQUÉS DE VALDECILLA UNIVERSITY HOSPITAL PACS WITH A CONVOLUTIONAL NEURAL NETWORK)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: Paula Criach Fernández

Director: Enrique Marqués Fraguela

Co-Director: Carlos Sainz Fernández

Junio - 2021

AGRADECIMIENTOS

Gracias a Enrique Marqués y Carlos Sainz por toda la ayuda y orientación que me han ofrecido durante la realización de este trabajo. Y a mis padres y hermana, por su eterna paciencia.

RESUMEN

En este trabajo se ha realizado la clasificación de 2470 imágenes médicas, extraídas del PACS del Hospital Universitario Marqués de Valdecilla, en 13 clases diferentes (parte del cuerpo y proyección radiográfica). Para ello, se ha utilizado la técnica de transferencia de aprendizaje, empleando como red neuronal convolucional preentrenada el modelo VGG-16 con los pesos de la competición de *imagenet*. El modelo se ha evaluado mediante la métrica *accuracy*, la matriz de confusión y los mapas de calor, sobre dos conjuntos de datos procedentes de equipos distintos. Para las imágenes del mismo equipo que las de entrenamiento se ha obtenido una exactitud del 93 %, mientras que para imágenes de equipo distinto se ha alcanzado el 66 %. Se ha logrado el objetivo principal, aunque no se puede concluir que la red sea capaz de generalizar a imágenes procedentes a otro equipo.

ABSTRACT

In this work, 2470 medical images, extracted from Marqués de Valdecilla University Hospital PACS, were classified into 13 different classes (part of the body and radiographic projection). For this purpose, the transfer learning technique has been used, using the VGG-16 model as the pretrained convolutional neural network with the weights of *imagenet* competition. The model has been evaluated on two datasetss, using the metric $textitaccuracy$, the confusion matrix and heat maps. For the dataset extracted from the training dataset, an accuracy of 93 % has been obtained, while for a different dataset consisting of images from a different equipment it has been reached 66 %. The main goal has been achieved, although it cannot be concluded that the network is capable of generalizing to images from another equipment.

Índice

1. Objetivo

La clasificación de imágenes médicas en un sistema de archivo no siempre es óptima y un porcentaje considerable de imágenes están catalogadas de forma errónea. Las razones por las que pueden existir problemas en el etiquetado de las imágenes son, la creación de estudios introduciendo los datos de forma manual, sin utilizar el mecanismo de la Lista de Trabajo y la realización de varios estudios como si fueran uno solo, sin cerrar el estudio previo y abrir uno nuevo, entre otras.

El Servicio de Radiofísica y Protección Radiológica, junto con los Servicios de Radiología e Informática, del Hospital Universitario Marqués de Valdecilla (HUMV) está tratando de impulsar el uso de la Inteligencia Artificial (AI) para mejorar la calidad de los servicios prestados a los pacientes en relación a la imagen médica. Una de las iniciativas que está impulsando consiste en la mejora de la clasificación de las imágenes médicas utilizando algoritmos de aprendizaje profundo (Deep Learning) como las redes neuronales convolucionales (CNNs).

El objetivo principal de este TFG es clasificar radiografías de forma automática en una serie de categorías (parte del cuerpo + proyección radiográfica) entrenando una red neuronal convolucional. El desarrollo de este TFG contribuirá a mejorar el etiquetado de las imágenes médicas que se almacenan en el PACS (Sistema de Comunicación y Archivado de Imágenes) del HUMV.

2. Introducción

2.1. Contexto Histórico

Muchos de los grandes descubrimientos científicos de la historia han sido predichos por la literatura de la ciencia ficción, el submarino de *"20.000 leguas de viaje submarino"* de Julio Verne, los auriculares en *"Farenheit 451"* de Ray Bradbury o incluso la bomba atómica en *"Un mundo se libera"* de H.G.Wells.

En la literatura del pasado se pueden encontrar descripciones del futuro muy acertadas en correspondencia con la realidad que vivimos en el presente, como dijo el filósofo Alejandro Piscitelli en un debate sobre la tecnología cotidiana: "La ciencia ficción nos ha permitido diseñar posibles mundos utópicos, nos propone un desbloqueo de la imaginación".

La Inteligencia Artificial lleva presente en este tipo de obras desde mediados del siglo XX, una de las más significativas e influyentes es el relato corto *"La Respuesta"* de Fredric Brown.

Las herramientas como la Inteligencia Artificial (IA), y en concreto, el Deep Learning están cambiando la forma en la que se encaran los problemas actuales de computación y están abriendo el abanico de las cosas a las que se pueden acceder gracias a un computador.

Poco después de la Segunda Guerra Mundial, en concreto a partir de 1956 se comenzó a hablar de la *Inteligencia Artificial*. Al tratarse de una disciplina nueva atrae a multitud de profesionales científicos de todas las disciplinas, esto es así porque aún tiene muchos "flecos sueltos" en los que poder trabajar. La *Inteligencia Artificial* automatiza tareas intelectuales normalmente realizadas por humanos, por lo que se considera potencialmente relevante en cualquier ámbito de trabajo.

La *Inteligencia Artificial* ha sufrido, a lo largo de toda su historia, periodos de fuerte optimismo seguidos de oleadas de pesimismo y falta de financiación, conocidas como inviernos de la *Inteligencia Artificial* (*AI winters*). A principios de la década de los 70, hubo dos sucesos que supusieron grandes contratiempos para el desarrollo de esta ciencia.

El informe *Lighthill*, en 1973 el Parlamento de Reino Unido evalúa el estado de la investigación sobre Inteligencia Artificial. En este informe se critica el fracaso de la IA a la hora de lograr sus supuestos grandiosos objetivos. Además, menciona que muchos de los algoritmos más exitosos de la IA fracasarían en problemas del mundo real y que solo eran adecuados para resolver versiones de "juguete"de los mismos. Este informe condujo al desmantelamiento completo de la investigación de IA en Inglaterra. La investigación de IA continuó en solo unas pocas universidades y esto provocó una oleada de recortes de fondos en toda Europa.

En 1960, la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA) proporcionó millones de dólares para la investigación en IA, casi sin condiciones. A partir de 1969, DARPA ya no financió la investigación pura no dirigida, sino que los investigadores debían demostrar que su trabajo produciría algún avance tecnológico con aplicaciones militares inmediatas. Esto, sumado al informe *Lighthill*, dió a entender que era poco probable que IA produjese algo realmente útil en un futuro inmediato, por lo que el dinero de DARPA se dirigió a otros proyectos.

Los algoritmos empleados en *Inteligencia Artificial* actualmente, mantienen las ideas primigenias de los años 60, con la gran diferencia de que ahora hay grandes empresas, como *Google* y *Facebook*,

invirtiendo en IA y se cuenta con mayor cantidad de datos y capacidad de computación.

Gracias a la IA se han producido grandes avances a lo largo de la historia, ELIZA fue uno de los primeros programas en procesar lenguaje natural en 1966. En 1996, el programa DEEP BLUE, ganó una partida de ajedrez al, en aquel entonces, campeón del mundo Gary Kaspárov. En 2012, las redes neuronales convolucionales ganan la competición de Imagenet clasificando un millón y medio de imágenes.

Dentro del contexto sanitario, la *Inteligencia Artificial* ha supuesto grandes avances tecnológicos, en especial en las técnicas de diagnóstico de enfermedades y en el análisis de imágenes médicas. Un claro ejemplo de la eficacia en este campo es su aplicación en los casos de cáncer de mama. Gracias a la IA se han desarrollado sistemas que permiten la detección del cáncer de mama en las etapas más tempranas de la enfermedad, superando en algunas ocasiones la capacidad de detección. Otro ejemplo en el que la IA tiene un papel crucial es en el desarrollo de prótesis inteligentes que memorizan los patrones de movimiento de la persona para adaptarse mejor a sus necesidades.

2.2. ¿Qué es la Inteligencia Artificial?

La Inteligencia Artificial se podría definir como el esfuerzo por automatizar las tareas intelectuales normalmente realizadas por humanos. Esta disciplina surgió en la década de 1950 a raíz de que la comunidad científica comenzase a preguntarse si los ordenadores podrían "pensar".

A lo largo de los años la Inteligencia Artificial ha sufrido un cambio de paradigma. Desde 1950 hasta 1980, la *Inteligencia Artificial* estaba guiada por el pensamiento humano. Los programadores debían establecer reglas explícitas para manipular los datos. Esto se conoce como IA simbólica.

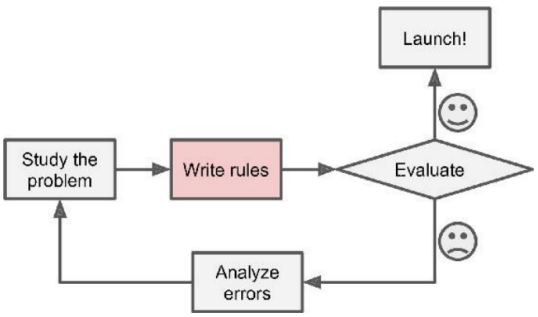

Figura 1: Enfoque de IA simbólica. (Fuente: [3])

Aunque este enfoque demostró ser adecuado para resolver problemas lógicos bien definidos, como una partida de ajedrez, resultaba insuficiente para resolver aquellos problemas en los que los humanos no pudiesen establecer reglas, como la clasificación de imágenes, el reconocimiento de voz o la traducción de idiomas. Debido a esta necesidad, surge un nuevo enfoque de la IA, el Aprendizaje Automático (*Machine Learning*).

El *Machine Learning* surge de la pregunta que en 1950 formuló Alan Turing acerca de si un ordenador podría aprender por si mismo y así, adquirir conocimiento. Los algoritmos de *Machine Learning* se programan para que sean capaces de aprender las reglas mediante la exposición a ejemplos resueltos. Una vez extraídas las reglas, estas se aplican a la resolución de problemas nuevos, lo que se denomina generalizar.

El *Machine Learning* es un subcampo de la Inteligencia Artificial que se basa en la búsqueda de métodos para dotar a las máquinas de la capacidad de aprender.

La propiedad más característica del ML frente a la IA simbólica, es que en el segundo caso para realizar la misma tarea, sería necesario que un experto encontrase y codificase las reglas que establecen la frontera de decisión.

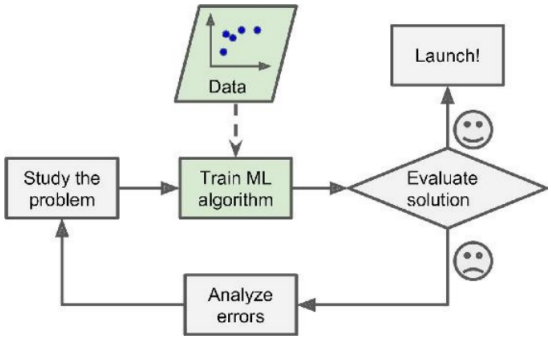

Figura 2: Enfoque de IA de Machine Learning. (Fuente: [3])

Los algoritmos de ML permiten resolver problemas para los que la solución requeriría una larga lista de reglas y aquellos que no se pueden resolver con IA simbólica. Son útiles para resolver problemas en ambientes cambiantes, ya que estos algoritmos se adaptan a nuevos datos.

2.3. Clasificación de algoritmos de Aprendizaje Automático

En función del grado de intervención del humano se pueden establecer tres grupos de algoritmos de *Machine Learning*: supervisado, no supervisado y por refuerzo.

• **No supervisado:** en este tipo de entrenamiento no es necesario disponer de datos etiquetados. Los sistemas son capaces de aprender patrones de similitud en los datos lo que permite su agrupación en diferentes categorías. Una de las ventajas de este tipo de algoritmos es que permiten aprender aspectos nuevos sobre el problemas. Las principales tareas que realizan estos algoritmos son de agrupación y asociación. Un ejemplo de este tipo de métodos serían el agrupamiento jerárquico (*(Hierarchical Clustering)*).

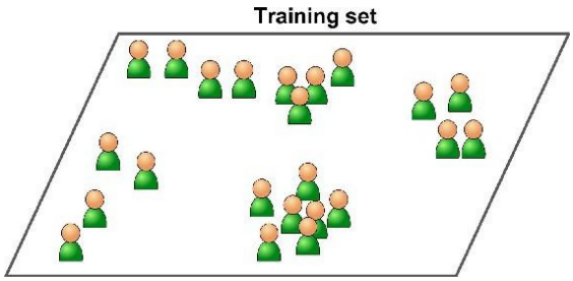

Figura 3: Esquema del conjunto de datos de entrenamiento en un algoritmo no supervisado. (Fuente: [3])

• **Supervisado:** en el aprendizaje supervisado, los datos de entrenamientos incluyen la solución a la tarea en forma de *etiqueta*.Estos algoritmos se aplican, sobretodo, en tareas de clasificación o

segmentación. Dado que los datos están etiquetados por un ser humano, el aprendizaje puede estar condicionado por dichas elecciones.

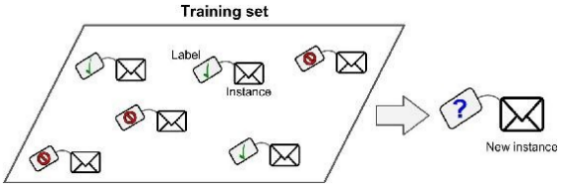

Figura 4: Esquema del conjunto de datos de entrenamiento en un algoritmo supervisado. (Fuente: [3])

• **Por refuerzo:** el sistema de aprendizaje llamado *agente* en este contexto, puede observar el entorno, seleccionar y realizar acciones, y obtener recompensas o sanciones a cambio. El *agente* debe aprender por sí mismo cuál es la mejor estrategia, llamada *política*, para obtener la mayor recompensa a lo largo del tiempo.

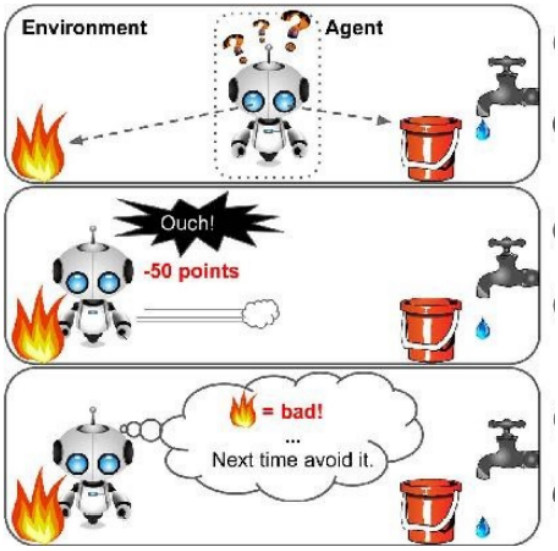

Figura 5: Aprendizaje por refuerzo. (Fuente: [3])

La mayoría de algoritmos de *Machine Learning* tienen cómo objetivo hacer predicciones. Obtener buenos predicciones con los datos de entrenamientos es necesario, pero no suficiente, el verdadero objetivo es que el sistema haga buenas predicciones con datos que no ha visto antes. Hay dos posibles enfoques en función de cómo representan el conocimiento: basados en instancias o en modelos.

• **Aprendizaje basado en instancias;** el sistema aprende los ejemplos de memoria y luego generaliza el conocimiento que ha aprendido a nuevos casos usando una medida de similitud. Por ejemplo, para crear una papelera de spam, el sistema se aprendería de memoria los correos previamente clasificados como “no deseados“ y enviaría a spam correos que contengan muchas palabras similares.

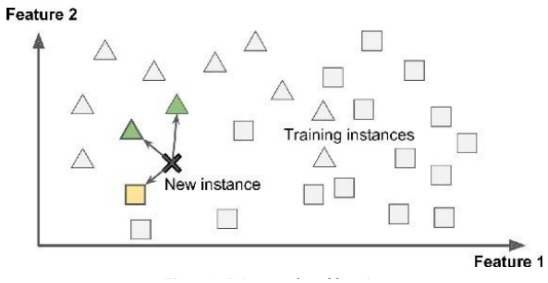

Figura 6: Aprendizaje basado en instancias. (Fuente: [3])

• **Aprendizaje basado en modelos;** otra manera de generalizar el conocimiento, es construir un modelo a partir de un conjunto de ejemplos y utilizarlo para hacer predicciones sobre nuevos datos.

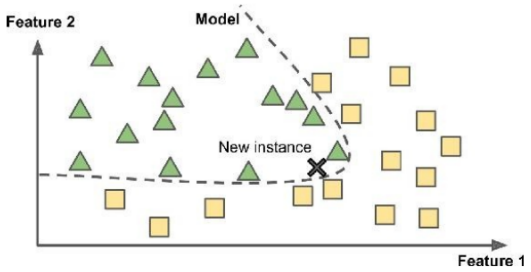

Figura 7: Aprendizaje basado en modelos. (Fuente: [3])

En función de la tarea que realizan, los algoritmos de *Machine Learning* se pueden dividir en algoritmos de clasificación, regresión, agrupación y asociación.

• **Algoritmos de clasificación:** se utilizan cuando la etiqueta toma valores discretos dentro de un conjunto finito de resultados. La clasificación puede ser binaria o múltiple.

• **Algoritmos de regresión:** su objetivo es establecer una relación entre un cierto número de características y una variable objetivo continua.

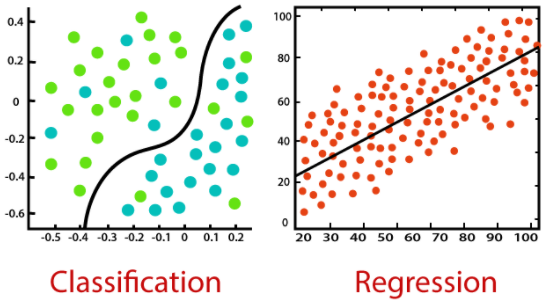

Figura 8: Esquema de algoritmos de clasificación y regresión. (Fuente:[13])

En función de su tarea, los algoritmos no supervisados se dividen en algoritmos de clustering y de asociación.

• **Algoritmos de agrupación (*clustering*):** es un procedimiento de agrupación de una serie de datos de acuerdo con un criterio, por lo general distancia o similitud entre casos.

• **Algoritmos de asociación:** los algoritmos de reglas de asociación tienen como objetivo encontrar relaciones dentro un conjunto de transacciones, en concreto, items o atributos que tienden a ocurrir de forma conjunta.

Algunos de los algoritmos más utilizados en *Machine Learning* son los siguientes:

Machine Learning			
Supervisado		No supervisado	
Clasificación	*Regresión*	*Clustering*	Asociación
Regresión logística	*Regresión lineal/no lineal*	*Númerico (K-medias)*	*A priori*
Árboles de decisión	*Máquinas de vector soporte*	*Conceptual (clustering jerárquico)*	
Bosques aleatorios	*Árboles de decisión*	*Estadístico (Expectation Maximation)*	
Redes neuronales	*Bosques aleatorios*		
Deep Learning	*Redes neuronales*		
	Deep Learning		

Figura 9: Clasificación de los algoritmos de Machine Learning más utilizados.

• **Métodos de regresión logística:** basados en la aplicación de los principios de la estadística al análisis de datos, Se utiliza para estimar la probabilidad de ocurrencia de un proceso en función de ciertas variables. Fue una de las primeras técnicas de *Machine Learning*, y es aún, muy utilizada a día de hoy. Es una técnica de tipo supervisada utilizada para resolver problemas de clasificación.

• **Árboles de decisión:** son estructuras en forma de diagrama de flujo que permiten clasificar en función de una serie de condiciones que ocurren de forma sucesiva. Existen técnicas que tratan de ensamblar varios árboles de decisión entrenados con diferentes muestras de los datos de entrenamiento, que reciben el nombre de bosques aleatorios. Es una técnica multidisciplinar, puede utilizarse para resolver problemas de clasificación y regresión.

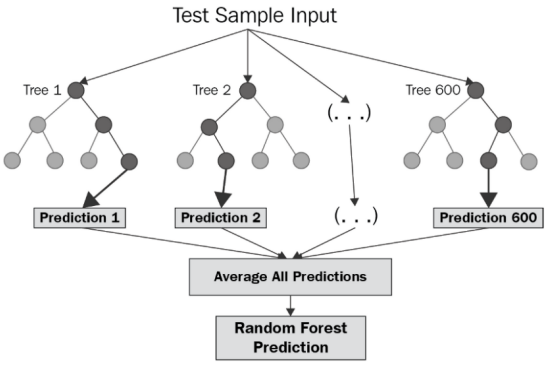

Figura 10: Estructura de bosque aleatorio. (Fuente: [11])

• **Clustering jerárquico:** es una técnica utilizada en algoritmos de agrupación de ML de tipo no supervisado, que sirve para determinar similitudes entre individuos acumulándolos en grupos (*clusters*).

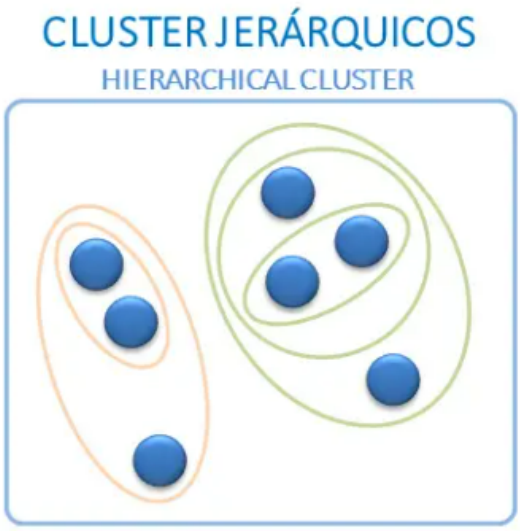

Figura 11: Agrupación de clustering jerárquico. (Fuente: [9])

• **K medias**: es un método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. La agrupación del conjunto de datos puede ilustrarse en una partición del espacio de datos en celdas de Voronoi.

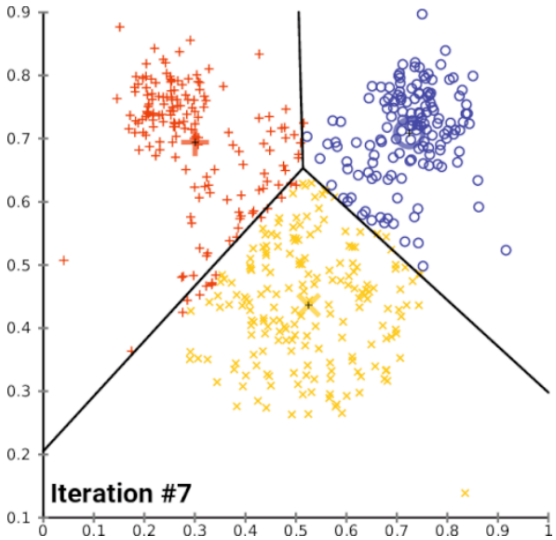

Figura 12: Agrupación con minería de datos con el método de K medias.(Fuente: [12])

• **Redes neuronales:**
Frank Rosenblatt creó en 1958 el *perceptrón*, la neurona artificial o unidad básica de inferencia en forma de discriminador lineal. Esta unidad elemental se inspira en las neuronas biológicas. Una red neuronal es una superposición de perceptrones conectados unos con otros en estructuras complejas. La estructura más sencilla se conforma de tres capas, una capa de entrada, una capa oculta, y una capa de salida. Cada entrada se pondera por separado y su suma se pasa a través de una función conocida como *función de activación*. Sin la cual la red neuronal sólo podría resolver problemas lineales. Estas funciones son equivalentes a las que determinan los potenciales umbral de activación de las neuronas biológicas.

En 1989 se realizó la primera aplicación práctica de una red neuronal terminó con éxito. *Bell Labs* junto con *Yann LeCun* combinaron las ideas originales de las redes convolucionales y la retropropagación en algoritmos de aprendizaje supervisado y las aplicaron a un problema de clasificación de dígitos escritos a mano.

Deep Learning es un caso particular de red neuronal que se caracteriza por tener múltiples capas ocultas, lo cual brinda un mejor desempeño que las redes de una sola capa oculta. El concepto subyacente en el aprendizaje profundo es el procesamiento de los datos de forma jerárquica. Es decir, el uso de redes neuronales para obtener representaciones cada vez más significativas de los datos mediante el aprendizaje por capas. Los modelos de *Deep Learning* modernos están formados por cientos de capas sucesivas que aprenden de forma automática por la exposición a datos. Estas capas se organizan en una capa de entrada, un conjunto de capas ocultas y una capa de salida

Cada capa extrae características de un nivel de abstracción cada vez más alto hasta llegar a su respuesta final. Al ir profundizando en la red, estas funciones más simples se van combinado para buscar relaciones más complejas como puedan ser partes concretas de la cara (ojos, boca, nariz). En un siguiente paso se identificaría la cara completa, y por último se identificaría a la persona a la que corresponde esa cara. Así las sucesivas capas de abstracción serían:

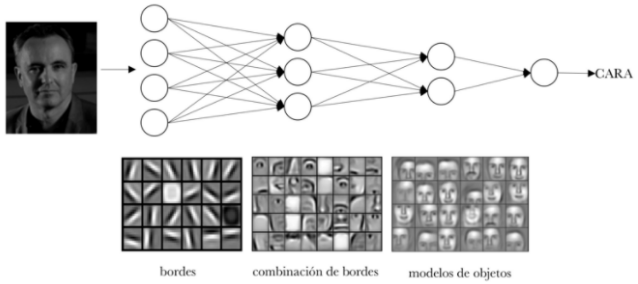

Figura 13: Esquema de funcionamiento de una red neuronal convolucional en el reconocimiento de una cara. (Fuente: [5])

Las redes neuronales más típicas en *Deep Learning* son las redes convolucionales, recurrentes, generativas, adversarias, etc.

3. Redes Neuronales

3.1. ¿Qué es una neurona?

Una neurona es un procesador elemental tal que a partir de unos datos de entrada procedentes del exterior o de otras neuronas, proporciona una única respuesta o salida.

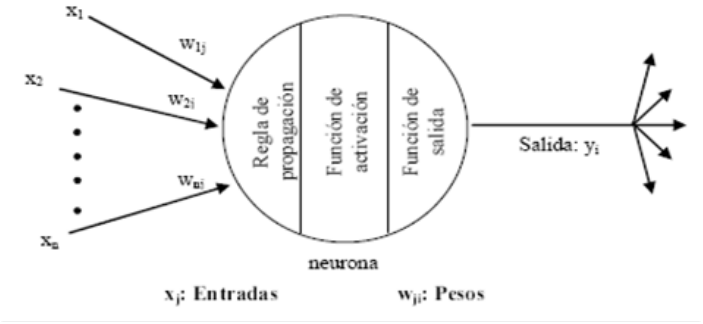

Figura 14: Modelo genérico de una neurona artificial. (Fuente: [15])

Los elementos básicos que constituyen una neurona son: la entrada x_i, los pesos W y sesgos b, la función de activación f y la salida o etiqueta y_i.

La neurona aplica el vector W de pesos calculado, ponderando sobre los elementos de entrada x_i, le suma el sesgo b, y el resultados se pasa a través de una función de activación lineal o no lineal para producir la salida (generalmente entre 0 y 1).

Las funciones de activación se utilizan para determinar el nivel de activación o inhibición de una neurona en función del resultado de multiplicar las entradas por los pesos y sumarles el sesgo.

$$z_i(t) = b + \Sigma_i x_i w_i \tag{1}$$

$$a_i(t) = f_i(z_i(t)) \tag{2}$$

Las funciones de activación más utilizadas son:

• **Función lineal:** es básicamente la función de identidad, donde la variable dependiente tiene una relación directa y proporcional con la variable independiente. En términos prácticos, significa que la función pasa la señal sin cambios.

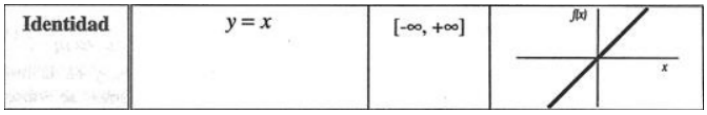

Figura 15: Función de activación lineal.(Fuente: [16])

• **Función sigmoide:** es aquella que convierte variables independientes de rango casi infinito en probabilidades simples entre 0 y 1. La mayor parte de su salida estará muy cerca de los entremos de 0 y 1.

• **Función tangente hiperbólica:** el rango normalizado de esta función está entre -1 y 1. La ventaja que nos ofrece está función es que puede tratar más fácilmente con números negativos.

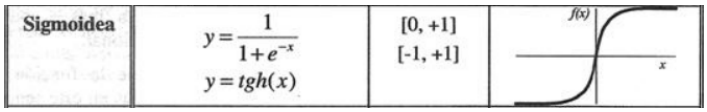

Figura 16: Función de activación sigmoide y tangente hiperbólica. (Fuente: [16])

• **Función Softmax:** en lugar de clasificar en binario, esta función puede contener múltiples límites de decisión. Devuelve la distribución de probabilidad sobre clases de salida mutuamente excluyentes. *Softmax* se utiliza a menudo en la capa de salida de un clasificador multiclase.

• **Función ReLu:** la función de activación *rectified linear unit* (ReLu) es una transformación que activa un solo nudo si la entrada está por encima de cierto umbral. El comportamiento más habitual es que mientras la entrada tenga un valor por debajo de cero, la salida sea cero, pero cuando la entrada se eleva por encima de cero, la salida es una relación lineal con la variable de entrada.

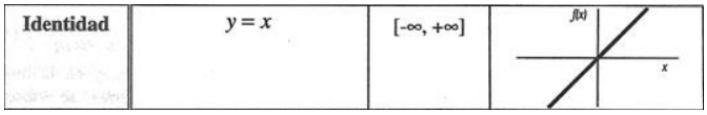

Figura 17: Función de activación ReLu.(Fuente: [16])

Los parámetros de la red se aprenden mediante un proceso iterativo, comparando el la predicción del modelo con la etiqueta real. Después de cada iteración, se ajustan los parámetros W y *b* de tal manera que se minimice la función de pérdida (*loss*).

La función de pérdida (*loss*) representa la penalización a una mala predicción. La pérdida es un número que indica cuán mala ha sido una predicción en un ejemplo concreto. Si la predicción fuese perfecta, la pérdida sería cero. Hay gran variedad de funciones de pérdida: *Mean square error*, *categorial cross entropy*, *binary cross entropy* o *sparse categorical cross entropy*. La elección de la mejor función de pérdida reside en entender qué tipo de error apropiado para el problema concreto.

• **Mean square error (MSE)**: es una función muy conocida que calcula el promedio de los cuadrados de los errores. Una variante de ésta sería la que, en lugar del cuadrado utiliza valor absoluto ('Absolute Error'). MSE se puede usar, por ejemplo, en problemas de regresión y con una última capa sin función de activación.

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(Y_i' - Y_i)^2 \tag{3}$$

Y$\Gamma esun vector den predicciones eY es el vector de valores verdaderos (etiquetas)$.

• **Categorial cross entropy**: la entropía cruzada, en general, es una medida de la distancia entre distribuciones de probabilidad. La entropía cruzada suele ser adecuada en modelos de redes cuya salida representa una probabilidad, como cuando hacemos una clasificación categórica con función de activación *Softmax*. Se utiliza cuando las etiquetas verdaderas están codificadas en *one hot encoding*.

$$L_{CE} = -\Sigma_{i=1} T_i log(S_i) \tag{4}$$

donde T_i es la etiqueta codificada y S_i es la probabilidad dada por la funció Softmax para la clase i.

• **Binary cross entropy**: es una variante de la anterior pero en la que tratamos con clasificación binaria y, por tanto, la función de activación es una *Sigmoide*.

$$L = -\frac{1}{N}(\Sigma_{j=1}^{N}[t_j log(p_j) + (1 - t_j) log(1 - p_j)]) \tag{5}$$

N número de datos donde t, es el valor verdadero (tomando valores de 0 y 1) y p_j la probabilidad para el dato i.

• **Sparse categorical cross entropy**: Es una variante que se suele usar en el caso de trabajar con muchas clases. En este caso la probabilidad predicha por la red para la mayoría de las clases es 0 y realizar operaciones innecesarias ralentiza el cálculo. Se utiliza esta función de pérdida para acelerarlo..

3.2. Arquitectura básica de las redes neuronales

La arquitectura de una red neuronal es la estructura en la que las distintas neuronas constituyentes de la red neuronal se asocian. Las neuronas se agrupan en unidades estructurales denominadas capas. Dentro de una misma capa, las neuronas suelen ser del mismo tipo. El conjunto de una o más capas constituye la red neuronal.

En una red neuronal artificial se puede distinguir tres tipos de capas:
• **Capa de entrada:** compuesta por neuronas que reciben datos o señales procedentes del entorno. El número de neuronas de esta capa deberá coincidir con el número de valores que representan cada muestra a analizar.
• **Capa de salida:** aquella cuyas neuronas proporcionan la respuesta de la red neuronal. En el caso de un clasificador, el número de neuronas debe coincidir con el número de clases.
• **Capa oculta:** aquellas que contienen unidades no observables y no tienen una conexión directa con el entorno. Es donde se realizan las operaciones matemáticas. El número de neuronas en este caso depende de la tarea que se realice.

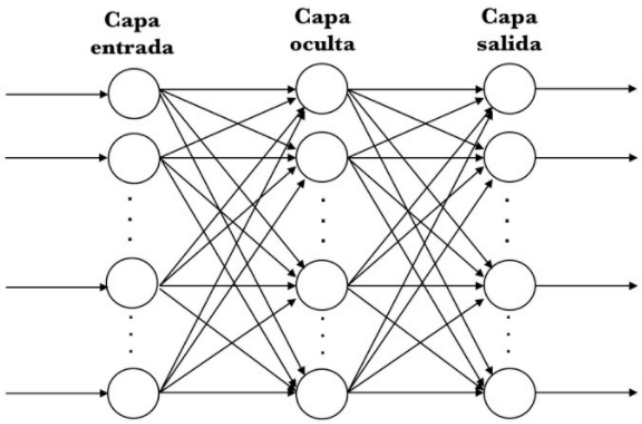

Figura 18: Arquitectura de capas de una red neuronal. (Fuente:[5])

Las conexiones pueden clasificarse en intracapa, entre las neuronas de una misma capa, e intercapa, entre neuronas de distintas capas.

Las capas densas son muy importantes en el desarrollo de redes neuronales. Estas son capas de cálculo que conectan cada neurona en una capa con todas las salidas de la capa anterior. Cuando una red contiene este tipo de capa se dice que está densamente conectadas.

Además, cuánto mayor es el número de capas ocultas se pueden resolver problemas más complejos. Estas redes con muchas capas ocultas se denominan redes neuronales profundas.

Una red neuronal convolucional (CNN) es un caso concreto de red neuronal profunda muy utilizada en problemas de visión artificial. Las redes neuronales convolucionales se han vuelto muy populares en los últimos años. Su rasgo diferencial es que las entradas de la red son imágenes y aprende una serie de filtros para reconocer elementos concretos en las imágenes.

En las CNN cada capa corresponde a un nivel de abstracción diferente, con redes de muchas capas se puede conseguir identificar estructuras muy complejas.

Los píxeles de las imágenes no son independientes los unos de los otros sino que su valor está muy ligado al de sus vecinos. Esto es lo que hace que surjan estructuras, formas y patrones que analizados correctamente permiten que la red entienda lo que ve. De esta idea nacen las redes neuronales convolucionales.

Las operaciones y tipos de capas que diferencian una CNN de cualquier otra red neuronal son: convolución y agrupación.

Las capas especializadas en operaciones de convolución, llamadas capas convolucionales, aprenden patrones locales en pequeñas ventanas de dos dimensiones. Su propósito principal es detectar características en las imágenes como aristas, gotas de color, etc. Lo interesante es que una vez aprendida la característica en un punto concreto de la imagen la puede reconocer en cualquier otro punto de la misma. Además, las capas convolucionales pueden aprender jerarquías espaciales, lo que permite que las CNN sean capaces de detectar patrones mucho más complejos. Estas capas operan sobre tensores de rango 3, llamados mapas de características, con dos ejes espaciales (altura y anchura) y un eje para representar el nivel de gris o color (profundidad o canal).

Un tensor es una clase de entidad algebraica de varios componentes que generaliza los conceptos de escalar, vector y matriz de una manera que sea independiente del sistema de coordenadas elegido.

Las CNN acompañan a las capas de convolución con unas capas de agrupamiento que suelen ser aplicadas inmediatamente después de estas. Las capas de agrupamiento hacen una simplificación de la información recogida por la capa convolucional y crean una versión condensada de esta. Las formas más utilizadas de condensar esta información son con la técnica *average pooling*, donde cada grupo de puntos de entrada se transforma en el valor promedio del grupo de puntos, o con la técnica *max pooling* donde se transforma en el valor máximo de los que había en la ventana de entrada.

Después de estas capas se utiliza una red densamente conectada que actúa como clasificador.

El diseño de una arquitectura convolucional se representa como un embudo, donde la imagen original se va comprimiendo espacialmente, su resolución va disminuyendo al mismo tiempo que el grosor va aumentando, es decir, que el número de mapas de características que se van detectando va en aumento. Cuando la imagen pasa por todo el embudo habrá detectado todos los patrones necesarios y contaremos con muchos mapas de características que se podrán meter como entrada independiente dentro de una red neuronal multicapa, el clasificador, que acabará por tomar la decisión de que es lo que estamos viendo.

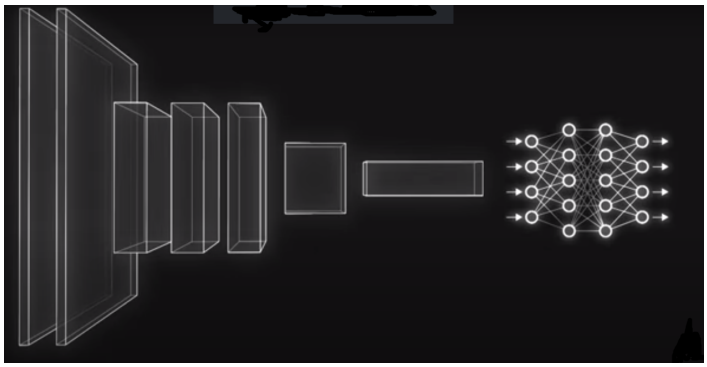

Figura 19: Esquema de la reducción de dimensionalidad en una CNN. (Fuente: [14])

3.3. Conjuntos de entrenamiento, evaluación y test

Existen tres conjuntos de datos fundamentales para el entrenamiento, selección y evaluación de un modelo, conjunto de entrenamiento, validación y test. El primero es el conjunto de datos con los que se entrena el modelo, el segundo es el conjunto de datos que se utiliza para seleccionar cuál es el mejor modelo y el tercero es el conjunto de datos con el que se evalúa el modelo elegido.

3.4. Entrenamiento

Entrenar una red consiste en ajustar los valores de los parámetros y es un proceso de propagación y retropropagación. Esto quiere decir que es un proceso iterativo de “ir y venir“ por las capas de neuronas.

El proceso de propagación consiste en que los datos de entrada pasan a través de la red de forma que las neuronas de las capas ocultas aplican transformaciones a la información que reciben de las capas anteriores y la envían a la capa siguiente, al llegar a la capa de salida, se llega a una predicción para los datos de entrada.

El siguiente paso es utilizar una función de pérdida para estimar el error y medir cómo de buena es la predicción en relación a la realidad. El objetivo es que el modelo tenga el mínimo error posible, es decir, que no exista diferencia entre la predicción y la realidad. A medida que se entrena el modelo, se van ajustando los pesos de las neuronas hasta obtener mejores predicciones.

Una vez obtenida la pérdida, se traspasa esa información hacia atrás. Partiendo de la capa de salida, la información de se propaga hacia todas las neuronas de las capas ocultas. Dichas neuronas no reciben toda esa información sino una parte proporcional basada en la contribución relativa que haya aportado cada neurona a la salida original.

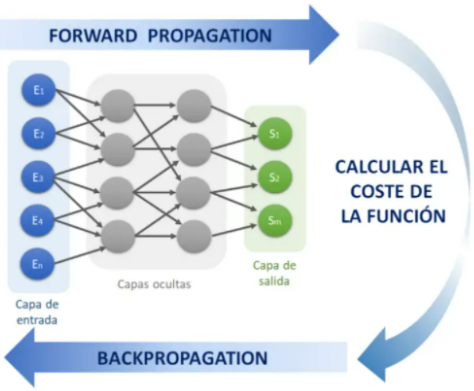

Figura 20: Esquema de funcionamiento de la retropropagación de información en una red neuronal. (Fuente: [10])

Una vez se ha propagado esa información hacia atrás se pueden ajustar los pesos de las conexiones entre neuronas. El objetivo es que la pérdida se aproxime lo más posible a cero la próxima vez que se haga una predicción.

Para ello se utiliza una técnica llamada descenso del gradiente. Esta técnica consiste en cambiar los pesos en pequeños pasos con ayuda del cálculo de la derivada de la función pérdida, lo que permite conocer en qué dirección "descender" hacia el mínimo global, esto se va haciendo en lotes de datos (*batches*) y en sucesivas iteraciones (*epochs*) del conjunto de datos. Para evitar que el optimizador se quede atascado en un mínimo local existe un hiperparámetro denominado *momentum*. Se puede entender como si un avance tomara el promedio ponderado de los pasos anteriores para obtener así un poco de ímpetu y superar los baches, como una forma de no atascarse en los mínimos locales. Este hiperparámetro es una constante que va de 0 a 1 y que realiza esta ponderación.

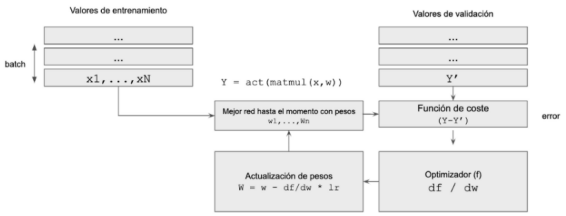

Figura 21: Algoritmo de retroprogación para entrenamiento de redes neuronales. (Fuente: [17])

El proceso de aprendizaje de una red se puede ver como un problema de optimización global, donde los parámetros se deben ajustar de tal manera que se minimice la función pérdida. El optimizador es el encargado de generar pesos cada vez mejores. Su funcionamiento esencial se basa en calcular el gradiente de la función de coste en función de los pesos de la red. Como se quiere minimizar la pérdida, se modifica cada peso en la dirección del gradiente. De cara a agilizar la convergencia de la función de coste hacia su mínimo, multiplicamos el vector de gradiente por un factor llamado tasa de aprendizaje (*learning rate*). En la mayoría de los casos, los parámetros se pueden ajustar bien con algoritmos de aproximación iterativos u optimizadores como: *Adagrad*, *Adadelta*, *RMSprop (Root Mean Square) Propagation)* o *Adam*.

- **Adagrad:** introduce una variación en el concepto de tasa de aprendizaje: en vez de considerar

un valor uniforme para todos los pesos, se adapta la tasa de aprendizaje a cada uno de ellos. Sería inviable calcular este valor de forma específica así que, partiendo del factor de aprendizaje inicial, AdaGrad lo escala y adapta para cada dimensión con respecto al gradiente acumulado en cada iteración.

• **Adadelta:** es una variación de AdaGrad en la que en vez de calcular el escalado de la tasa de aprendizaje de cada dimensión teniendo en cuenta el gradiente acumulado desde el principio de la ejecución, se restringe a una ventana de tamaño fijo de los últimos n gradientes.

• **RMSprop:** mantiene una tasa de aprendizaje diferente para cada dimensión, pero en este caso el escalado del la tasa de aprendizaje se realiza dividiéndolo por la media del declive exponencial del cuadrado de los gradientes.

• **Adam:** combina las ventajas de AdaGrad y RMSProp. Se utiliza una tasa de aprendizaje por parámetro y además de calcular RMSProp, cada tasa de aprendizaje también se adapta por la media del momentum del gradiente.

Por hiperparámetro nos referimos a variables de configuración que son externas al modelo y cuyos valores no pueden ser ajustados a partir de los datos sino que son especificados por el programador para optimizar los algoritmos de aprendizaje. Los hiperparámetros más importantes de una red neuronal son:

• **Número de épocas (*epochs*:)** indica el número de veces que van a pasar por la red los datos de entrenamiento. Aumentar el número de épocas puede hacer que el aprendizaje mejore.

• **Tamaño de lote (*Batch size*):** es el hiperparámetro que indica el número de datos que se usará en cada iteración para actualizar el gradiente. El tamaño óptimo depende de muchos factores, como por ejemplo la memoria del computador que se utilice para hacer los cálculos.

• **Tasa de aprendizaje (*learning rate*):** los algoritmos del *descenso del gradiente* multiplican la magnitud del gradiente por un escalar conocido como *learning rate* para determinar el siguiente punto. Por ejemplo, si la magnitud del vector gradiente es 1,5 y el *learning rate* 0,01, entonces el algoritmo del *descenso del gradiente* seleccionará el siguiente punto a 0,015 del anterior. El valor adecuado de dicho parámetro depende mucho del problema en cuestión, si es demasiado grande, dará pasos enormes y aunque el proceso de aprendizaje será más rápido puede saltarse los mínimos de la función de coste y no converger. Por el contrario, si se escoge uno demasiado pequeño, encontrará mejor los mínimos de la función de coste, pero el proceso será muy lento.

3.5. Evaluación

La técnica de *Hold out* consiste en dividir el conjunto de datos en dos, un conjunto de entrenamiento y otro conjunto de test.

La tasa de error en casos nuevos se denomina error de generalización y evaluando el modelo sobre el conjunto de test se puede obtener una estimación. Este valor permite conocer cómo de bueno es el modelo haciendo predicciones sobre casos que nunca ha visto.

Si el error en el entrenamiento es bajo, pero el error de generalización es alto, significa que el modelo está en sobreajuste (*overfitting*).

Una forma de mejorar la generalización y así evitar el sobreajuste, es entrenar el modelo con distintos hiperparámetros y ajustarlos hasta producir el menor error de generalización posible.

Se entrenan múltiples modelos con diferentes hiperparámetros usando el conjunto de entrenamiento, y se selecciona el conjunto que mejor funciona sobre el conjunto de validación. Una vez se ha decidido cuál es el mejor modelo, se hace una prueba final sobre el conjunto de test (*nunca jamás* visto por el modelo) y se estima el *error de generalización*.

Para evaluar cómo de buenas son las predicciones de nuestro modelo respecto de la realidad se utiliza la métrica. Algunos ejemplos de métricas de evaluación de modelos de clasificación son los siguientes:

• **Exactitud (*Accuracy*)**: es la fracción de predicciones que el modelo realizó correctamente. Se representa como un porcentaje o un valor entre 0 y 1, cuando más cercano a 1 es el valor mejor comportamiento tiene la red.

• **Sensibilidad (*Recall*)**: indica la proporción de ejemplos positivos que están identificados correctamente por el modelo entre todos los positivos reales.

• **Precisión**: esta métrica está determinada por la fracción de elementos clasificados correctamente como positivo entre todos los que el modelo ha clasificado como positivos.

• **Matriz de confusión:** es una herramienta ampliamente utilizada que permite inspeccionar y evaluar visualmente las predicciones de nuestro modelo. En las filas se representan las predicciones del modelo y en las columnas las etiquetas que representan la realidad o la verdad. La diagonal principal se corresponden con los valores estimados de forma correcta por el modelo. El resto de diagonales por lo tanto, representan los casos en los que el modelo ha clasificado la entrada de forma errónea.

Figura 22: Matriz de confusión. (Fuente: [8])

Otro elemento crucial para ganar confianza sobre las predicciones de la red son los mapas de calor. Gracias a esta herramienta se puede visualizar lo que valora la red neuronal convolucional cuando hace una predicción, permite acceder a las características que está tomando nuestro modelo como determinantes para clasificar las imágenes.

Para construir el mapa de calor, se ha utilizado una técnica llamada Grad-CAM (Mapa de activación de clases basado en el gradiente). La idea básica es que para encontrar la importancia de una determinada clase en nuestro modelo, tomamos su gradiente con respecto a la capa convolucional final y lo comparamos con la salida de esta capa.

4. Material y métodos

El problema a resolver consistió en clasificar imágenes médicas del Hospital Universitario Marqués de Valdecilla en 13 clases diferentes (parte del cuerpo y proyección radiográfica).

Para ello se ha utilizado la técnica de transferencia de aprendizaje utilizando una red preentrenada para extraer las características de las imágenes. Esta técnica consiste en tomar las características aprendidas en un problema ya resuelto y aprovecharlas en el nuestro. Hemos utilizado esta técnica porque no contamos con un conjunto de datos lo suficientemente grande como para entrenar un modelo desde cero. Para ello, utilizamos como red neuronal convolucional preentrenada el modelo VGG-16 con los pesos de la competición de *imagenet* para extraer las características de las imágenes necesarias para entrenar el clasificador. Este modelo está compuesto por un total de 19 capas, una de entrada, 13 de convolución y 5 de agrupamiento. Hemos congelado las capas de este modelo para evitar destruir la información que contienen durante las futuras rondas de entrenamiento y hemos añadido unas capas nuevas y entrenables sobre las congeladas que actuarán como clasificador.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 512, 512, 3)	0
block1_conv1 (Conv2D)	(None, 512, 512, 64)	1792
block1_conv2 (Conv2D)	(None, 512, 512, 64)	36928
block1_pool (MaxPooling2D)	(None, 256, 256, 64)	0
block2_conv1 (Conv2D)	(None, 256, 256, 128)	73856
block2_conv2 (Conv2D)	(None, 256, 256, 128)	147584
block2_pool (MaxPooling2D)	(None, 128, 128, 128)	0
block3_conv1 (Conv2D)	(None, 128, 128, 256)	295168
block3_conv2 (Conv2D)	(None, 128, 128, 256)	590080
block3_conv3 (Conv2D)	(None, 128, 128, 256)	590080
block3_pool (MaxPooling2D)	(None, 64, 64, 256)	0
block4_conv1 (Conv2D)	(None, 64, 64, 512)	1180160
block4_conv2 (Conv2D)	(None, 64, 64, 512)	2359808
block4_conv3 (Conv2D)	(None, 64, 64, 512)	2359808
block4_pool (MaxPooling2D)	(None, 32, 32, 512)	0
block5_conv1 (Conv2D)	(None, 32, 32, 512)	2359808
block5_conv2 (Conv2D)	(None, 32, 32, 512)	2359808
block5_conv3 (Conv2D)	(None, 32, 32, 512)	2359808
block5_pool (MaxPooling2D)	(None, 16, 16, 512)	0

Figura 23: VGG-16.

Las capas que hemos añadido son las siguientes:

• **Global Average Pooling:** es una operación de agrupación diseñada para reducir la dimensionalidad. En lugar de agregar capas densamente conectadas en la parte superior de los mapas de características, tomamos el promedio de cada mapa de características.

• **Dense:** utilizamos una capa densamente conectada formada por 256 neuronas con una función de activación ReLu.

• **Dropout:** es un método que desactiva un número determinado de neuronas en una red neuronal de forma aleatoria. Las neuronas desactivadas no se tienen en cuenta para la propagación hacia delante

ni para atrás, lo que obliga a las neuronas cercanas a no depender tanto de las neuronas desactivadas. Este hecho ayuda a reducir el sobreajuste. *Dropout* tiene un parámetro que indica la probabilidad de que las neuronas se queden o no activadas y que toma valores de 0 a 1. Hemos establecido un parámetro de 0,5.

• **Dense:** esta capa densa actúa como capa de salida y está formada por tantas neuronas como clases (13) con una función de activación tipo *softmax*.

```
base_model = applications.VGG16(include_top=False, weights='imagenet',input_shape=(512, 512, 3))
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D(input_shape=base_model.output_shape[1:],data_format='channels_last'))
model.add(Dense(256, kernel_regularizer = regularizers.l1_l2(l1=0.001, l2=0.001),activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(13, activation='softmax'))
base_model.trainable=False
```

Figura 24: Arquitectura de la red neuronal convolucional.

4.1. Material

4.1.1. Recursos computacionales

Para la realización de este proyecto he accedido en remoto a un computador del Instituto de Física de Cantabria (IFCA) perteneciente al Consejo Superior de Investigaciones Científicas (CSIC).

CPU:
Modelo: Intel® Xeon® Gold 6230
Frecuencia de reloj: 2,10 GHz típica, hasta 3,90 GHz con la tecnología Intel® Turbo Boost
Hilos de procesamiento: 32
Consumo: 125W

Memoria RAM:64 Gb DDR4-2933

GPU:
Modelo:NVIDIA Tesla V100
Núcleos tensoriales: 640
Núcleos CUDA: 5120
Capacidad de procesamiento: 112 TFLOPS
Memoria RAM: 32 Gb HBM2
Consumo: 250 W

4.1.2. Entorno de trabajo

Se utilizó como entorno de programación *Jupyter Notebooks*, que es una aplicación cliente-servidor que permite crear y compartir documentos en formato JSON que siguen un esquema versionado y una lista ordenada de celdas de entrada y de salida. Estas celdas albergan, entre otras cosas, código, texto, fórmulas matemáticas y ecuaciones, o también contenido multimedia. El programa se ejecuta desde la aplicación web que funciona en cualquier navegador estándar. El requisito previo es instalar y ejecutar en el sistema el servidor Jupyter Notebook. Los documentos creados en Jupyter pueden exportarse, entre otros formatos, a HTML, PDF, Markdown o Python.

Las librerías utilizadas en este trabajo para implementar las redes neuronales fueron TensorFlow y Keras. Tensorflow es una librería de implementación de redes neuronales de Google y Keras una

librería de alto nivel diseñada para facilitar el uso de Tensorflow y de otras librerías de construcción de redes neuronales.

Actualmente Keras está incluido en Tensorflow y además, se puede utilizar como una librería de Python.

Para sacarle el máximo partido a los *Jupyter Notebooks* se instaló como núcleo de ejecución Python. Además, se utilizaron las siguientes librerías de Python:

• **Numpy:** Es la librería de cálculo numérico de Python que soporta el manejo de tensores, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellos.

• **Scikit-learn:** Es una biblioteca de aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión, agrupación, etc.

• **Matplotlib:** Es la librería de visualización más popular escrita en Python.

• **Pydicom:** Es un paquete de Python para trabajar con archivos DICOM, como son las imágenes médicas que hemos utilizado en este trabajo. Pydicom facilita el leer estos complejos archivos en estructuras de python para una sencilla manipulación.

• **Os:** El módulo os de Python nos permite realizar operaciones dependientes del Sistema Operativo como crear una carpeta, listar contenidos de una carpeta, o navegar a través de los directorios.

• **CV2:** Es una biblioteca libre utilizada para el desarrollo de aplicaciones de visión artificial.

4.1.3. Conjunto de datos

Las imágenes se extrajeron del PACS (Picture Archiving and Communication System) del Hospital Universitario Marqués de Valdecilla y se anonimizaron antes de su uso. Se utilizaron imágenes en formato DICOM y se procesaron antes de introducirlas a la red.

Como se realizó un entrenamiento de tipo supervisado, las imágenes utilizadas fueron previamente revisadas y etiquetadas. En el proceso de limpieza del conjunto de datos de entrenamiento se observó que las imágenes estaban marcadas por unos símbolos que indicaban su lateralidad y la posición del paciente. Además, se observó que dicha marca estaba colocada en un cuadrante distinto en cada imagen.

Para que la red neuronal no considerase estas etiquetas, o su posición, como un patrón característico de una clase de imagen médica concreta y esto, condujese a predicciones erróneas, se realizó un proceso previo de elección y organización de las imágenes de entrenamiento. Para cada clase, se escogieron imágenes representativas de todos los casos posibles para que la red no aprendiese un patrón concreto que luego no generalizaría a datos nuevos.

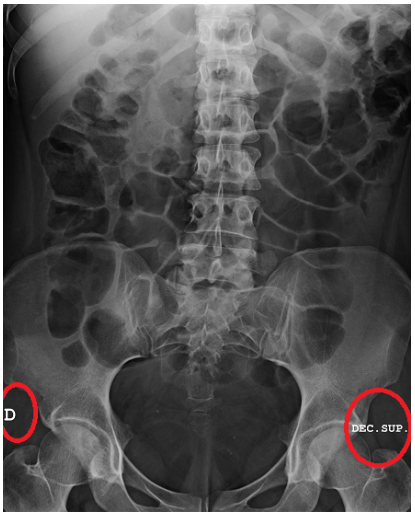

Figura 25: Imagen ABDOMEN SIMPLE AP, derecha (lateralidad D) y en posición de decúbito supino (DEC. SUP)

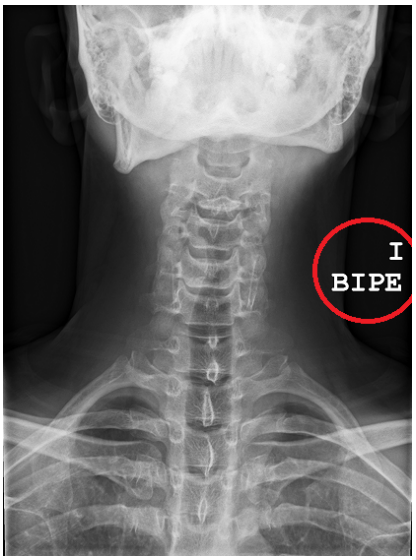

Figura 26: Imagen COLUMNA CERVICAL AP, izquierda (lateralidad I) y en posición de bidepestación (BIPE).

En primer lugar,se invirtieron los valores de píxel de las imágenes, para ello se restó el valor del píxel máximo del de cada píxel. Se aplicó una ventana, en nuestro caso se utilizó una ventana completa que cubre todos los valores posibles de 0 a 4096 (12 bits). Dado que las imágenes eran muy grandes, se redujo su tamaño a 512x512 píxeles. La red que se utilizó solo soportaba como datos de entrada imágenes en color por lo que se convirtieron las imágenes en escala de grises a imágenes en color (RGB). Por último, se normalizaron las imágenes dividiendo el valor de cada píxel por el valor máximo (4096) de forma que los valores de píxel se situaban en el intervalo entre 0 y 1.

```
pixel_array= ds.pixel_array
largest_image_pixel_value= np.power(2,ds.BitsStored)
pixel_array_inverted= largest_image_pixel_value - pixel_array
pixel_array_inverted = pixel_array
ds.WindowCenter= largest_image_pixel_value/2
ds.WindowWidth= largest_image_pixel_value
pixel_array_windowed= handlers.apply_windowing(pixel_array_inverted,ds)
pixel_array_resized= cv2.resize(pixel_array_windowed,(512,512),interpolation=cv2.INTER_CUBIC)
pixel_array_normalized= pixel_array_resized/np.max(pixel_array_resized)
pixel_array_rgb= np.stack([pixel_array_normalized, pixel_array_normalized, pixel_array_normalized], axis=-1)
```

Figura 27: Código para procesar las imágenes

Tras procesar las imágenes de entrenamiento, se obtuvo un tensor de rango 4, (2081, 512, 512, 3). El primer elemento es el número de imágenes, los dos siguientes son el número de píxeles (512x512) y el último es el número de canales de color (3).

Además, se construyó un tensor de rango 2 con las etiquetas (*labels*) de las imágenes, (2081, 13). El primer elemento es el número de imágenes y el segundo el número de clases. En lugar de utilizar números enteros 0,1,2,3...13 se utilizazó la codificación *one hot encoding* para transformar los números enteros en tensores de rango 1 ortogonales (1000000000000), (01000000000000), (0010000000000), etc.

A continuación se presentan las etiquetas que se utilizaron para cada clase y una imagen representativa de cada una.

Clase	Etiqueta(*label*)
Abdomen antero-posterior	[1 0 0 0 0 0 0 0 0 0 0 0 0 0]
Pie antero-posterior	[0 1 0 0 0 0 0 0 0 0 0 0 0 0]
Rodilla antero-posterior	[0 0 1 0 0 0 0 0 0 0 0 0 0 0]
Tobillo antero-posterior	[0 0 0 1 0 0 0 0 0 0 0 0 0 0]
Columna cervical antero-posterior	[0 0 0 0 1 0 0 0 0 0 0 0 0 0]
Columna lumbosacra antero-posterior	[0 0 0 0 0 1 0 0 0 0 0 0 0 0]
Pie antero-posterior oblicuo	[0 0 0 0 0 0 1 0 0 0 0 0 0 0]
Tórax lateral	[0 0 0 0 0 0 0 1 0 0 0 0 0 0]
Rodilla lateral	[0 0 0 0 0 0 0 0 1 0 0 0 0 0]
Tobillo lateral	[0 0 0 0 0 0 0 0 0 1 0 0 0 0]
Columna cervical lateral	[0 0 0 0 0 0 0 0 0 0 1 0 0 0]
Columna lumbosacra lateral	[0 0 0 0 0 0 0 0 0 0 0 1 0 0]
Tórax postero-anterior	[0 0 0 0 0 0 0 0 0 0 0 0 1 0]

Tabla 1: Etiquetas de las trece clases de imágenes médicas.

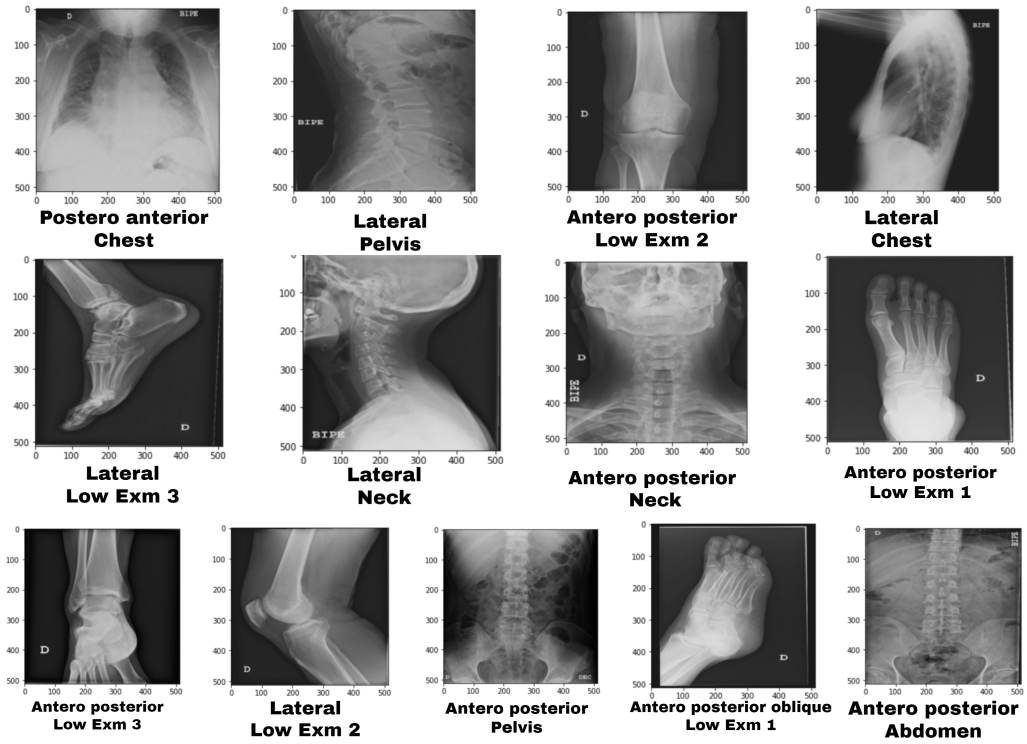

Figura 28: Imágenes representativas de cada una de las clases.

Hemos contado con un total de 2470 imágenes. Se dividieron los datos en dos conjuntos, el conjunto de entrenamiento con 2080 imágenes (160 imágenes de cada clase) y el conjunto de test con 429 imágenes (33 de cada clase). Además, se utilizó un segundo conjunto de test con 297 imágenes procedentes de un equipo distinto al de entrenamiento. El conjunto de entrenamiento se dividió en dos conjuntos para escoger el mejor modelo, el 75 % (1560 imágenes) para entrenar y el 25 % (520 imágenes) para validar.

4.2. Métodos

4.2.1. Entrenamiento y evaluación de la red

Para definir el modelo se utilizó la función de activación *softmax* ya que es la más óptima para clasificadores múltiples con pocas clases, como nuestro caso. En cuanto al optimizador, se eligió el *ADAM* (Adaptive Moment Estimation) ya que combina las ventajas del Adagrad y de RMSprop, además requiere poca memoria, y es apropiado para este tipo de problema. La tasa de aprendizaje se estableció en 0,001. Al tratarse de un clasificador múltiple y porque las etiquetas se codificaron en *one hot encoding*, la función de pérdida que se empleó fue *categorical cross entropy*. Por último, en cuanto a la métrica se estableció *accuracy*, es la medida más directa e intuitiva de la exactitud de los clasificadores.

```
model.compile(optimizer=Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-09, decay=0.0), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figura 29: Método para definir el modelo con las características establecidas.

Para ajustar los hiperparámetros se hizo una búsqueda exploratoria por el espacio de los hiperparámetros. Se escogió como modelo aquel que presentaba mejor métrica sobre el conjunto de validación. Finalmente, se tomaron como tamaño de lote (*batch size*) = 32 y épocas *epochs* = 200.

```
history=model.fit(training_data, y_label_training, batch_size=32, epochs=200, verbose=1, validation_data = (validation_data, y_label_validation))
```

Figura 30: Entrenamiento y validación para elegir el mejor modelo.

El siguiente paso fue evaluar y realizar las predicciones del modelo sobre el conjunto de datos de test. Para ello se utilizaron dos conjuntos de datos, uno precedente del mismo equipo que las imágenes con las que se ha entrenado, denominada validación interna y otro conjunto de un equipo distinto para estudiar su capacidad de generalización, validación externa. Gracias a la métrica, se observó cómo de bien funciona la red.

```
score = model.evaluate(X_test_array, y_label_test_array_one_hot_encoding, verbose=0)
```

Figura 31: Evaluación.

```
pred = model.predict(X_test_array, batch_size = None, verbose = 0, steps = None, callbacks = None, max_queue_size = 10, workers = 1, use_multiprocessing = False)
y_pred = np.argmax(pred)
```

Figura 32: Predicción.

Las dos herramientas que se utilizaron para visualizar el aprendizaje del modelo y para ganar confianza sobre las predicciones obtenidas fueron la matriz de confusión y los mapas de calor.

```python
from sklearn.metrics import confusion_matrix, f1_score, roc_curve, precision_score, recall_score, accuracy_score, roc_auc_score
#from sklearn import metrics
#from mlxtend.plotting import plot_confusion_matrix
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt

y_pred = np.argmax(pred)
y_real = y_label_test_array

matc=  confusion_matrix(y_real, y_pred)
plt.imshow(matc,cmap='gray')

# For visualization purpose, we will also normalize the heatmap between 0 & 1
matc = np.maximum(matc, 0) / np.max(matc)

matc = cv2.resize(matc,(512,512))
matc = (matc*255).astype("uint8")
matc = cv2.applyColorMap(matc,cv2.COLORMAP_JET)

#For saving the heatmap
cv2.imwrite(os.path.join('matriz_confusion.jpg'), matc)

np.save('matriz confusion', matc)
```

Figura 33: Obtención de la matriz de confusión a partir de las predicciones.

```python
def make_gradcam_heatmap(img_instance, model, last_conv_layer_name, pred_index=None):

    base_model_input_object = keras.Input(shape=img_instance.shape[1:])

    base_model = model.get_layer('vgg16')
    #base_model.summary()
    x = base_model_input_object
    base_model_layer_names= [layer.name for layer in base_model.layers]
    for layer_name in base_model_layer_names:
        x = base_model.get_layer(layer_name)(x)
    base_model_object = keras.Model(base_model_input_object, x)

    base_model_object_output = base_model_object.output
    classifier_object_input = keras.Input(shape=base_model_object_output.shape[1:])
    x = classifier_object_input
    model_layer_names= [layer.name for layer in model.layers]
    classifier_layer_names= model_layer_names[1:]
    for layer_name in classifier_layer_names:
        x = model.get_layer(layer_name)(x)
    classifier_model_object = keras.Model(classifier_object_input, x)

    with tf.GradientTape() as tape:
         # Compute activations of the base model and make the tape watch it
        base_model_output_instance = base_model_object(img_instance)
        tape.watch(base_model_output_instance)
        # Compute class predictions
        preds = classifier_model_object(base_model_output_instance)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]

    # This is the gradient of the top predicted class with regard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, base_model_output_instance)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    base_model_output_instance = base_model_output_instance.numpy()[0]
    pooled_grads = pooled_grads.numpy()
    for grad in range(pooled_grads.shape[-1]):
        base_model_output_instance[:, :, grad] *= pooled_grads[grad]

    # The channel-wise mean of the resulting feature map
    # is our heatmap of class activation
    heatmap = np.mean(base_model_output_instance, axis=-1)

    return heatmap
```

Figura 34: Código para generar el mapa de calor y superponerlo a la imagen

5. Resultados

5.1. Validación interna

• **Métrica:**

La exactitud obtenida en la evaluación de dicho conjunto de datos es del 93,85 %

```
4s 9ms/sample - loss: 0.7490 - accuracy: 0.9324
```

Figura 35: Métrica obtenida en la validación interna.

• **Matriz de confusión:**

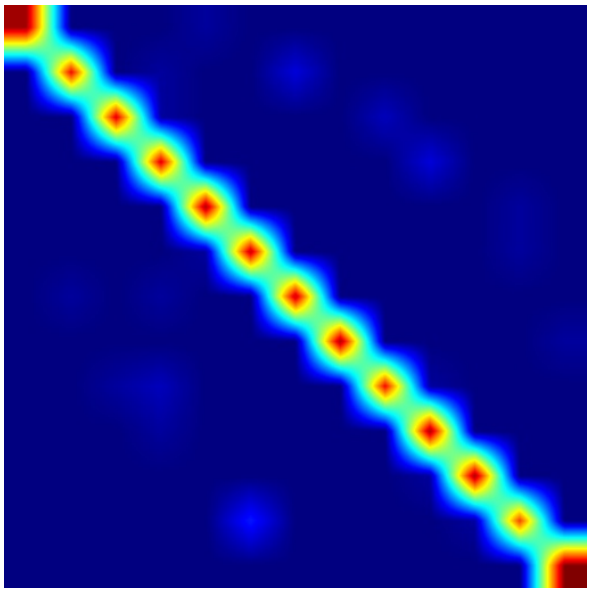

Figura 36: Matriz de confusión del conjunto de evaluación

• **Mapas de calor:**

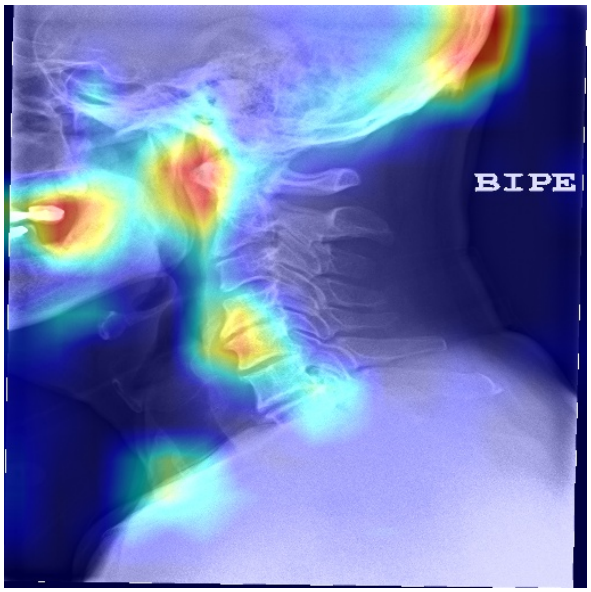

Figura 37: Mapa de calor representativo de las imágenes clasificadas como columna cervical en proyección lateral.

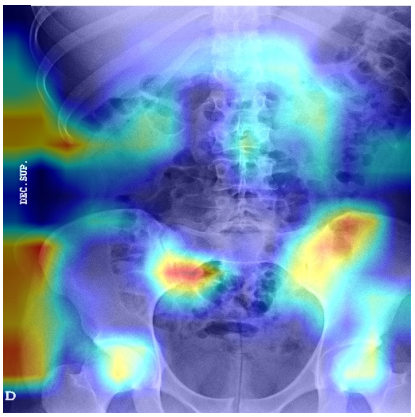

Figura 38: Mapa de calor representativo de las imágenes clasificadas como abdomen simple en proyección antero-posterior.

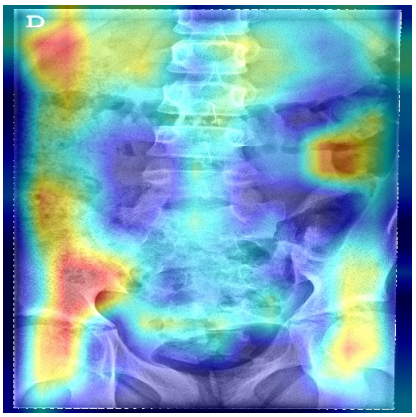

Figura 39: Mapa de calor representativo de las imágenes clasificadas como columna lumbosacra en proyección antero-posterior.

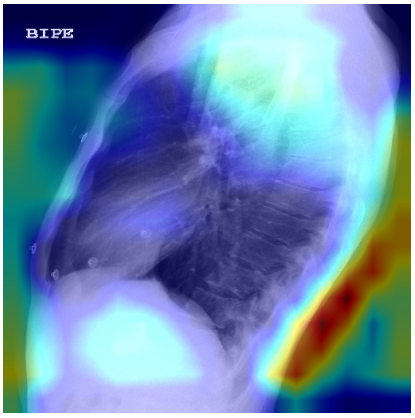

Figura 40: Mapa de calor representativo de las imágenes clasificadas como tórax en proyección lateral.

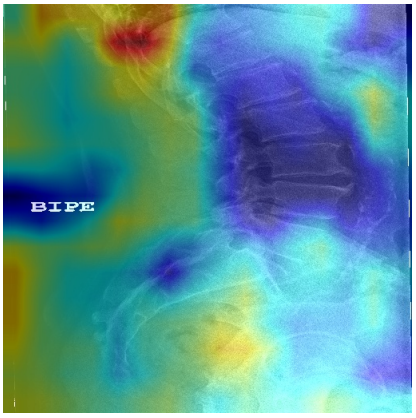

Figura 41: Mapa de calor representativo de las imágenes clasificadas como columna lumbosacra en proyección lateral.

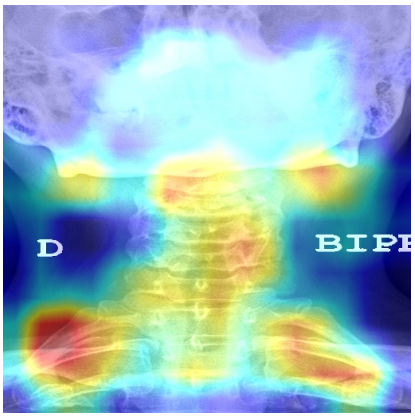

Figura 42: Mapa de calor representativo de las imágenes clasificadas como columna cervical en proyección antero-posterior.

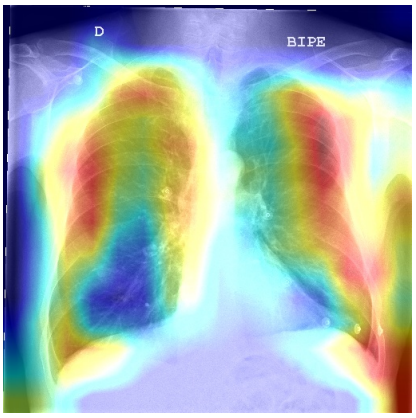

Figura 43: Mapa de calor representativo de las imágenes clasificadas como tórax en proyección postero-anterior.

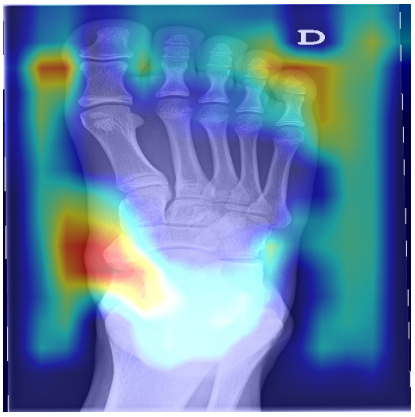

Figura 44: Mapa de calor representativo de las imágenes clasificadas como pie en proyección antero-posterior.

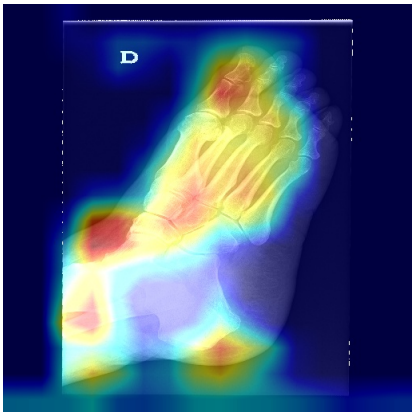

Figura 45: Mapa de calor representativo de las imágenes clasificadas como pie en proyección antero-posterior oblicuo.

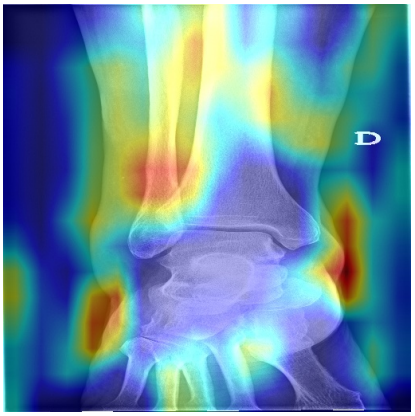

Figura 46: Mapa de calor representativo de las imágenes clasificadas como tobillo en proyección antero-posterior.

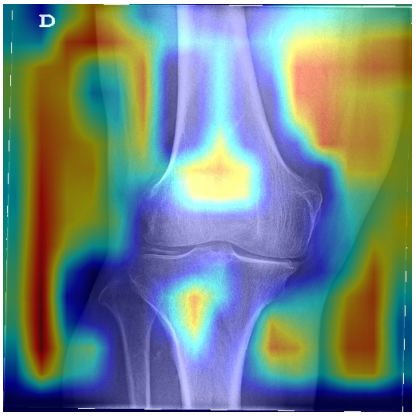

Figura 47: Mapa de calor representativo de las imágenes clasificadas como rodilla en proyección antero-posterior.

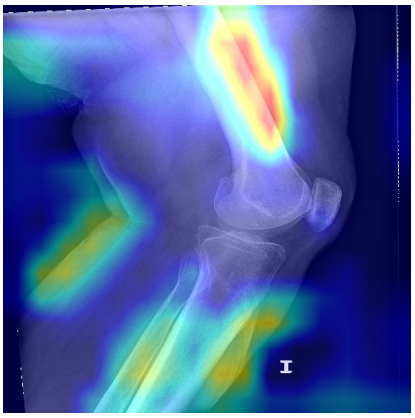

Figura 48: Mapa de calor representativo de las imágenes clasificadas como rodilla en proyección lateral.

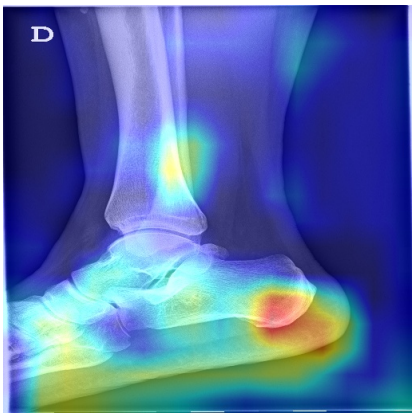

Figura 49: Mapa de calor representativo de las imágenes clasificadas como tobillo en proyección lateral.

5.2. Validación externa

• **Métrica:**

La exactitud obtenida en la evaluación de dicho conjunto de datos es del 66,20 %

```
11s 25ms/sample - loss: 2.2061 - accuracy: 0.6620
```

Figura 50: Métrica obtenida en la evaluación del conjunto de datos de un equipo distinto que el conjunto de entrenamiento.

• **Matriz de confusión:**

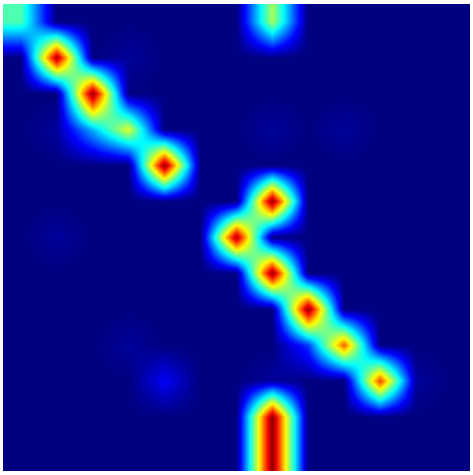

Figura 51: Matriz de confusión.

• **Mapas de calor:**

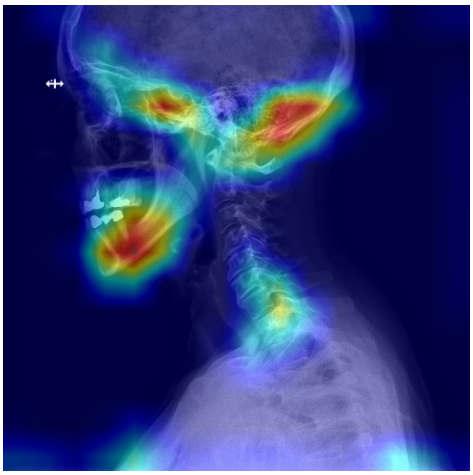

Figura 52: Mapa de calor representativo de las imágenes clasificadas como columna cervical en proyección lateral.

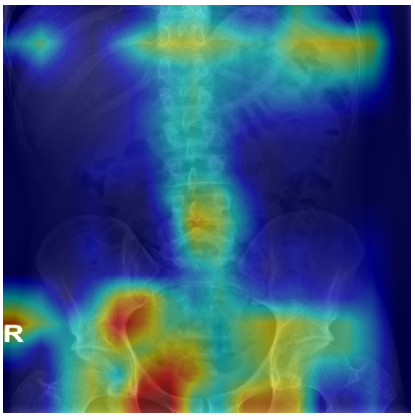

Figura 53: Mapa de calor representativo de las imágenes clasificadas como abdomen simple en proyección antero-posterior.

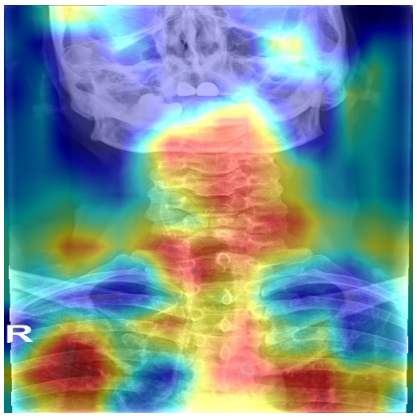

Figura 54: Mapa de calor representativo de las imágenes clasificadas como columna cervical en proyección antero-posterior.

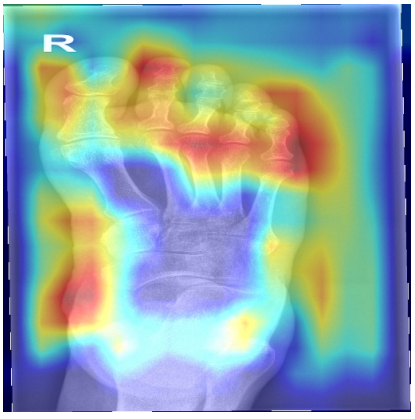

Figura 55: Mapa de calor representativo de las imágenes clasificadas como pie en proyección antero-posterior.

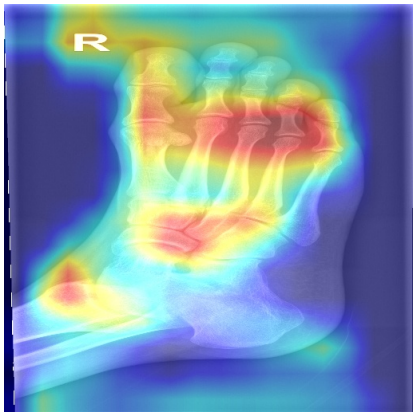

Figura 56: Mapa de calor representativo de las imágenes clasificadas como pie en proyección antero-posterior oblicuo.

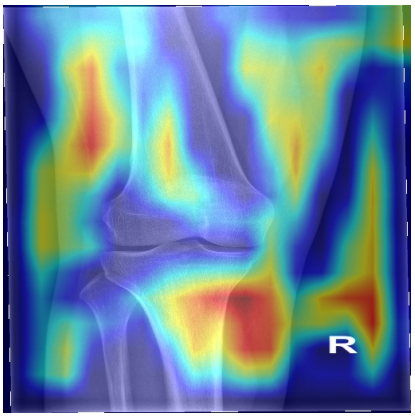

Figura 57: Mapa de calor representativo de las imágenes clasificadas como rodilla en proyección antero-posterior.

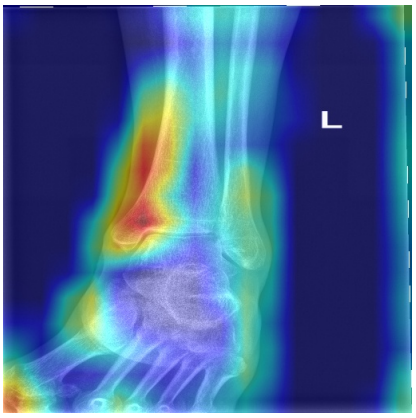

Figura 58: Mapa de calor representativo de las imágenes clasificadas como tobillol en proyección antero-posterior.

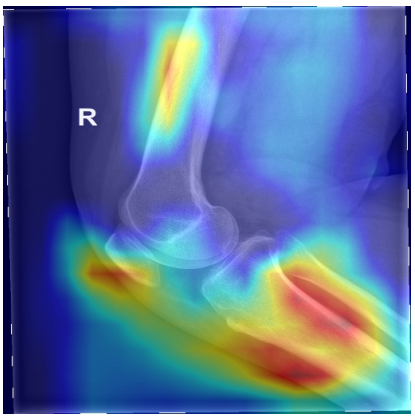

Figura 59: Mapa de calor representativo de las imágenes clasificadas como rodilla en proyección lateral.

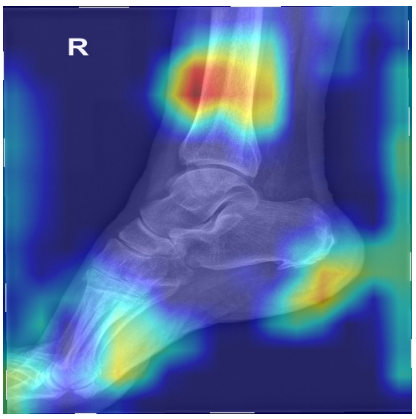

Figura 60: Mapa de calor representativo de las imágenes clasificadas como tobillo en proyección lateral.

6. Discusión y conclusiones

6.1. Validación interna

Los resultados obtenidos en validación interna son los esperados en un problema de clasificación múltiple, como el que nos ocupa.

Hemos obtenido una exactitud del 93 % por lo que podemos concluir que el modelo clasifica correctamente las imágenes en parte del cuerpo y proyección radiográfica.

La matriz de confusión nos permite observar gráficamente la eficacia del modelo, pues prácticamente todas las imágenes caen sobre la diagonal, excepto casos puntuales. Estos casos aislados, que han sido mal clasificados, pueden deberse a anomalías en las imágenes u otras causas.

Los mapas de calor nos han permitido abrir la supuesta "caja negra" de las redes neuronales. Hemos podido observar cómo la red no clasifica las imágenes de forma aleatoria ni en función de los marcadores como temíamos desde un principio que podía ocurrir. La red clasifica las imágenes siguiendo un patrón aprendido.

Una curiosidad observada al estudiar los mapas de calor es que la red se fija en características distintas que las que un principio utilizaría previsiblemente un humano. Al clasificar las extremidades podemos ver cómo apenas se fija en los huesos, muy similares en pie, rodilla y tobillo, sino que se fija en los límites entre músculo y aire.

Para distinguir entre las imágenes clasificadas como abdomen simple antero-posterior o columna lumbosacra antero-posterior, dos proyecciones cuyas imágenes son muy similares, observamos que en las últimas da mayor importancia a la columna vertebral y huesos de la pelvis. En el caso de las imágenes de tórax antero-posterior, se fija sobre todo en la cavidad pulmonar.

En cualquier caso, esto nos permite confiar en las predicciones realizadas por la red, pero no podemos asegurar que la interpretación que le estamos dando sea certera.

6.2. Validación externa

Con el objetivo de comprobar que el modelo generaliza de manera correcta,lo hemos evaluado con imágenes procedentes de otro equipo.

Al tratarse de un equipo distinto, las imágenes tienen características diferentes, en este caso, por ejemplo, no hemos tenido que invertir los píxeles en el procesamiento. Esto nos hizo prever que las predicciones no iban a ser tan buenas como en el caso anterior. La métrica que obtuvimos nos terminó por dar la razón, con una exactitud del 66 %.

En este caso, la matriz de confusión no es tan clara sino que hay muchos puntos que han sido clasificados de forma errónea. Si bien es cierto, que de este equipo no contábamos con imágenes de todas las clases, en principio eso no tendría que afectar a que el resto sean correctamente clasificadas.

En cuanto a los mapas de calor, lo más importante es que en este caso no evita los marcadores de

posición. Más bien lo contrario, en la mayoría de los casos están muy destacados. Gracias a esta herramienta de visualización de los resultados, podemos apreciar que aunque el 66 % de las imágenes han sido bien clasificadas, la red no sigue un patrón preciso como para garantizar la confianza en el modelo en este caso.

6.3. Conclusión

En general podemos concluir que los resultados obtenidos han sido los previstos desde el principio. Obtenemos unos resultados realmente buenos cuando evaluamos la red con imágenes procedentes del mismo equipo, pero insuficientes cuando introducimos imágenes de un equipo distinto.

Esto supone que el modelo que se ha creado no tenga una utilidad real para implementarlo en un hospital ya que es necesario que la red clasifique las imágenes independientemente del equipo al que pertenezcan.

El buen o mal funcionamiento de un modelo depende de dos factores, el algoritmo o los datos. Por lo que estos dos aspectos son las que se pueden perfeccionar para obtener mejores predicciones.

• **Datos:** los algoritmos de *Machine Learning* necesitan una gran cantidad de datos para funcionar correctamente. Para que sea capaz de generalizar con nuevos datos, es importante entrenar el modelo con aquellos que contengan toda la información necesaria, es decir, que sean representativos. Si los datos de entrenamiento están llenos de errores, valores atípicos y ruido, será más difícil para el sistema detectar los patrones, por lo que es menos probable que funcione bien.

Para que el modelo generalice a imágenes de distintos equipos, tendríamos que entrenarlo con imágenes de ambos. Además, estas imágenes tendrían que estar balanceadas, es decir, que haya el mismo nivel de representación de todas las clases y de ambos equipos. Al igual que hemos hecho con nuestro modelo, merecerá la pena invertir tiempo en hacer limpieza de los datos y entrenar el modelo con los más representativos.

• **Algoritmo:** los grandes problemas a los que nos podemos enfrentar cuando hablamos de un algoritmo de *Machine Learning* son el sobreajuste e infrajuste. Se dice que un modelo está en sobreajuste si está empezando a aprender patrones muy específicos de los datos de entrenamiento, pero equivocados o irrelevantes en datos nuevos. El infrajuste (*underfitting*) es el proceso opuesto al sobreajuste *overfitting*. Ocurre cuando un modelo es muy simplista, insuficiente para capturar los matices, particularidades y complejidades en los datos de entrenamiento.

En nuestro caso, hemos solventado los problemas de sobreajuste o infrajuste del modelo durante la elección de los mejores hiperparámetros. Si volviésemos a entrenar el modelo con mayor número de imágenes, y con datos de los dos equipos, tendríamos que volver a ajustarlos para encontrar el modelo más óptimo.

En general, podemos concluir que los resultados obtenidos son correctos pues hemos alcanzado el objetivo principal, aunque hay margen de mejora para trabajar en un futuro. Con más tiempo y recursos, los resultados y la generalización del modelo pueden mejorar sensiblemente haciendo nuestra red mucho más útil para una situación real.

7. Bibliografía

[1] BBC News Mundo. (2015, 14 septiembre). Los 10 hitos más importantes en la historia de la inteligencia artificial. https://www.bbc.com/mundo/noticias/2015/09/150914tecnologiainteligenciaartificial hitos10turingasimovamv

[2] Chollet, F. (2017). Deep Learning with Python. Manning Publications.

[3] Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. Van Duuren Media.

[4] Pianykh, O. S. (2011). Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide (2nd ed.). Springer.

[5] Torres, J. (2018). DEEP LEARNING Introducción práctica con Keras (WATCH THIS SPACE) (Spanish Edition). Independently published.

[6] IDIVAL Instituto de Investigación. (2021, 12 mayo). Curso Tendencias en Investigación Clínica: Sesión 13 [Vídeo]. YouTube. https://www.youtube.com/watch?v=dUqObSaEmRg

[7] Webedia Brand Services. (2019, 15 febrero). Los principales hitos en la historia de la inteligencia artificial. Ecosistema Huawei. https://ecosistemahuawei.xataka.com/principales-hitos-historia-inteligencia-artificial/

• Bibliografía web:

[8] Arce, J. I. B. (2021, 15 junio). La matriz de confusión y sus métricas. Juan Barrios. https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

[9] Calvo, D. (2018, 1 diciembre). Clúster Jerárquicos y No Jerárquicos. Diego Calvo. https://www.diegocalvo.es/cluster-jerarquicos-y-no-jerarquicos/

[10] Calvo, D. (2019, 19 agosto). Definición de red neuronal artificial. Diego Calvo. https://www.diegocalvo.es/definicion-de-red-neuronal/

[11] Chakure, A. (2020, 6 noviembre). Random Forest Regression - The Startup. Medium. https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f

[12] D. (2019, 11 marzo). K-Means: Agrupamiento con Minería de datos [Introducción]. ESTRATEGIAS DE TRADING. https://estrategiastrading.com/k-means/

[13] D. (2021, 11 junio). Algoritmos Supervisados: Clasificación vs. Regresión – Datlas Research. Blog de www.datlas.mx. https://blogdatlas.wordpress.com/2020/06/28/algoritmos-supervisados-clasificacion-vs-regresion-datlas-research/

[14] Dot CSV. (2020, 12 noviembre). ¡APRENDE Qué son las Redes Neuronales CONVOLUCIONALES! [Vídeo]. YouTube. https://www.youtube.com/watch?v=V8j1oENVz00

[15] ELEMENTOS BÁSICOS DE UNA RED NEURONAL ARTIFICIAL. (2007, 1 octubre). Advanced Tech Computing Group UTPL. https://advancedtech.wordpress.com/2007/08/31/elementos-basicos-de-una-red-neuronal-artificial/

[16] Sánchez Anzola, N. (2015). Máquinas de soporte vectorial y redes neuronales artificiales en la predicción del movimiento USD/COP spot intradiario. ODEON, 9, pp. 113-172.

[17] Velasco, L. (2020, 4 septiembre). Optimizadores en redes neuronales profundas: un enfoque práctico. Medium. https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5